Patterns of Mobile Interaction

JÖRG ROTH

University of Hagen, 58084 Hagen, Germany

Joerg.Roth@Fernuni-hagen.de
Tel. +49 2331 987 2134
Fax. +49 2331 987 390

Abstract. Designing systems for mobile scenarios covers a wide range of issues, ranging from mobile networking to user interface design for mobile devices. Mobile applications often run distributed on several connected devices, used by many users simultaneously. Considering all issues related to mobile scenarios, a designer might be overwhelmed. As a solution, we propose a specific kind of design patterns we call *mobility patterns*. Mobility patterns have been derived from successful mobile applications and allow a designer to reuse design elements as building blocks. After describing the idea of mobility patterns, we give a brief overview of some patterns we identified so far.

Keywords: mobile computing, mobile interaction, application design, design patterns

Introduction

Mobile computing fundamentally differs from desktop computing. Mobile devices, e.g. PDAs, mobile phones and digital cameras have, compared to desktop computers low computational power, small memory and often no mass storage. Communication links to other mobile devices or to a stationary network are usually wireless, thus often unstable with low bandwidth.
These aspects highly influence how users interact with mobile systems. Often, users interacting with a mobile system interact with other users (using, e.g., mobile phones). In contrast to traditional applications, users often interact with more than one computer or device at the same time. Having, e.g., a PDA browsing the Internet, a user may have to set up a wireless connection using a mobile phone.

Patterns

Having several users and several devices in a system design, a designer has to consider several problem areas. To give some examples:

- A single application has to be developed in a distributed manner, i.e. parts have to be identified which run independently on different devices.
- Due to the poor availability, bandwidth and reliability of mobile connections, we have to address network problems.

- Security (e.g., privacy, integrity and authenticity) is an important issue, since data in mobile scenarios are often confidential data [7].
- Designing user interfaces for mobile devices, especially for heterogeneous environments, we have to consider the special user requirements as well as the capabilities of the involved devices [2,7].

Often, a design focuses on a specific problem area and neglects others. Especially in the complex field of mobile computing, neglecting problems can cause an entire design to fail. On the other hand, taken into account all issues in early design stages may overwhelm a designer or a design team.
As a possible solution, we propose a tool successfully used in different scenarios: *design patterns*. Patterns are derived from successful software designs and can be reused as building blocks for new designs. In our approach, we use a specific kind of design patterns we call *mobility patterns*. Mobility patterns cover problem areas we find very often in mobile scenarios. Related patterns can be grouped together to *pattern classes* using a pattern hierarchy (figure 1). The white boxes represent the patterns, where the grey boxes represent pattern classes.

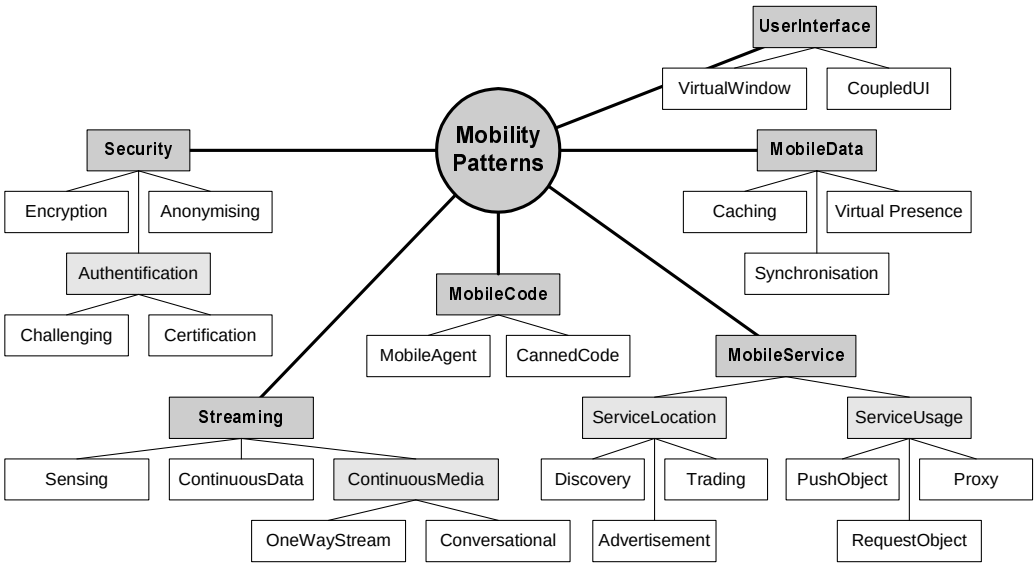

Fig 1. The pattern hierarchy

Mobility patterns are not only applicable for mobile scenarios but some pattern appear very often in mobile application projects, thus are good candidates for new projects.

Example Patterns

To describe patterns, we followed established description formats (e.g. [3]). Each description contains the sections: *Pattern Name*, *Synopsis*, *Context*, *Forces*, *Solu-*

tion, *Consequences*, *Examples*, *Related Patterns* and *Classes*. From the proposed description used in object-oriented software development, we replaced the section *Implementation* by *Examples*. It is difficult to give a tangible implementation for a specific mobility pattern. Instead, a list of good examples is more meaningful. In addition we added the section *Classes*, which indicates, how a pattern is integrated into the pattern hierarchy. Pattern classes give a designer additional structuring information, thus problems and solutions related to a specific pattern can be classified more easily. Compared to more formal approaches (e.g. [1]), mobility patterns have a strongly informal characteristic. To give an impression of mobility patterns, we present two patterns: *Synchronisation* and *Proxy*. Note that in this paper, we can only give a brief summary of the patterns. Original pattern descriptions need several pages.

Example 1: The Synchronisation Pattern

Synopsis: Identical data is stored on different devices or computers, which are weakly connected. Several users apply data changes to different devices, often simultaneously.

Context: Consider two users carrying around databases stored in their PDAs, which have been originally copied from a central server. While travelling, they are only rarely connected to their home database.

Forces: Identical data stored at different locations tends to run *out of sync* when users apply modifications on their locally stored data and device are only rarely connected.

Solution: Each device or computer provides a *sync engine* which

- keeps track of modifications applied to local data;
- exchanges modifications with corresponding databases on other devices whenever they reconnect;
- detects conflicts and realises a conflict resolution strategy.

Consequences: This pattern can be used whenever data stored on different devices has not to be strongly up to date. However, users should be aware that other users could change the same data simultaneously, which may cause conflicts later. Since connected devices have to effectively compare their local changes, only data, which can be stored in tables or lists, can be used with this pattern. Ideally, changes are logged for every field in a record. Weakly structured data can be compared record by record, which increases the probability for unwanted conflicts. This pattern is not suitable for continuous data such as audio or video.

Examples:

- SyncML (http://www.syncml.org) is a framework for data synchronisation in mobile scenarios.

- Palms Hotsync allows a user to synchronise a palm device with one ore more desktop PCs.
- QuickStep [7, 8] synchronises data between mobile devices and stationary servers, as well as between stationary servers among themselves.

Related Patterns: VirtualPresence
Classes: MobileData

Example 2: The Proxy Pattern

Synopsis: A device has not the capability to perform a requested task. It connects to another device with higher computational power, which acts as a delegate.
Context: Consider a user browsing the web with a handheld device. The screen resolution of such a device is currently very poor, thus, elements such as graphics and tables are difficult to display. In addition, rendering complex elements requires many computational resources, which are often unavailable on such a device.
Forces: A user who requests a specific task

- wants to safe network bandwidth,
- wants to safe computational resources (e.g. memory) on the local device,
- expects appropriate input/output behaviour according to the locally available capabilities.

Solution: The device does not connect directly to the required service end point, but asks another device or computer to perform these tasks. This other computer, called the *proxy*,

- accepts service requests from other devices,
- connects to the actual service provider and performs the requested tasks,
- processes the results and
- sends them back to the initiating device.

Consequences: This pattern is a general pattern and useful in various mobile scenarios. Very often, the devices used by the end-users have only poor capabilities. Nevertheless, users want to execute demanding tasks. In this pattern, there are two crucial points:

- the proxy itself: in case of failure, tasks execution is disabled, even if the requesting device and the service provider are on-line;
- the communication link between the requesting device and proxy: if this link is broken, the task cannot be performed, even if the proxy has successfully executed the task.

Obviously, the proxy has to have more or other capabilities than the end-user's device. As the proxy provides a specific service, a mobile device must be able to find the proxy inside the network. This leads to the following consequences:

- A proxy has to have a fixed network address or the mobile device can, with help of a service discovery mechanism, resolve the network address.
- The proxy has to be on-line whenever a mobile devices requests services.

Currently, a proxy is usually a traditional workstation, not a mobile device.

Examples:

- ProxyWeb (http://www.intellisync.com) allows a handheld user to browse the web without struggling with device limitations. The proxy pre-processes web pages, downscales graphics and pre-computes the appropriate layout. As a result, the amount of data transferred to the handheld devices is drastically reduced and the devices are relieved from heavy rendering tasks.
- PocketDreamTeam is the PalmOS version of the groupware platform DreamTeam [6]. Since PalmOS does not provide some services necessary to run the full DreamTeam platform, PocketDreamTeam uses a proxy, which performs specific tasks. As a result, the PocketDreamTeam program is very small (some Kbytes) compared to the original DreamTeam platform (some hundreds Kbytes).

Related Patterns: PushObject, RequestObject

Classes: ServiceUsage, MobileService

More Mobility Patterns

Table 1 lists some more mobility patterns we identified so far. Due to the limited space, only the brief synopsis and an overview of examples are presented.

Table 1. A list of more mobility patterns

Pattern	Synopsis	Examples
VirtualPresence	Pieces of data are virtually present to other devices and can be accessed or modified according to the host's access rules.	Windows CE remote file access, Coda [4]
RequestObject	One device requests a specific object (e.g. a web page) from another device.	WAP
PushObject	One device sends a specific object without request.	SMS, OBEX
OneWayStream	One-way transmission of audio or video data.	Video on demand, UMTS
Conversational	Two-way transmission of audio or video data.	GSM, DECT, Bluetooth audio
VirtualWindow	One device presenting a window or desktop of another device or computer.	Pebbles [5], PalmVNC
CannedCode	One device sends code, which is executed on another device. The code has not to be executable on the sender's device.	WMLscript, web filters
Sensing	One device receives continuous sensor data (e.g. location) from other devices.	GPS

Benefits

Using mobility patterns has many advantages. Primarily, patterns are a tool to describe designs. Using the proposed pattern names, a designer can precisely express which building blocks are used for a specific system, thus misunderstanding is more unlikely.
As described above, mobile applications cover a wide range of issues. Designers tend to 'abstract away' or forget consequences of specific design aspects, thus possibly build systems which are not runnable. As a second benefit, patterns come along with a list of implications and consequences. A designer is clear about the pros and cons of a specific pattern.
As a third benefit, patterns allow a designer to reuse successful designs. Reusing software on lower levels, with e.g., software components, fail due to the heterogeneity of involved networks and devices. With mobility patterns, at least the design of a successful application can be reused.
The current pattern hierarchy does not claim to be complete. In the future we want to complete our collection of mobility patterns. For this, we analyse existing mobile computing applications and frameworks.

References

[1] Borchers J.: A pattern approach to interaction design, Conference proceedings on Designing interactive systems: processes, practices, methods, and techniques Aug. 17 - 19 2000, Brooklyn, NY United States, 369-378

[2] Calvary G., Coutaz J., Thevenin D.:A Unifying Reference Framework for the Development of Plastic User Interfaces, 8th IFIP Working Conference on Engineering for Human-Computer Interaction (EHCI'01), Toronto, Canada, May 11-13 2001, in press

[3] Gamma E., Helm R., Johnson R., Vlissides J., Design Patterns: Elements of Reusable Object-Oriented Software, Addison Wesley, 1995

[4] Kistler J. J., Satyanarayana M.: Disconnected Operation in the Coda File System, ACM Transaction on Computer Systems, Vol. 10, No. 1, Feb. 1992, 3-25

[5] Myers B. A., Stiel H., Gargiulo R.: Collaboration Using Multiple PDAs Connected to a PC, Proceedings of the ACM 1998 conference on Computer supported cooperative work, 1998, 285-294

[6] Roth J.: DreamTeam - A Platform for Synchronous Collaborative Applications, AI & Society (2000) Vol. 14, No. 1, Springer London, March 2000, 98-119

[7] Roth J., Unger C.: Using handheld devices in synchronous collaborative scenarios, Second International Symposion on Handheld and Ubiquitous Computing 2000 (HUC2K), Bristol (UK), Sept. 25-27 2000, LNCS 1927, Springer, 187-199

[8] Roth J.: Information sharing with handheld appliances, 8th IFIP Working Conference on Engineering for Human-Computer Interaction (EHCI'01), Toronto, Canada, May 11-13 2001, in press